Mr. Giansante

Visual Basic

Arrays

August 2016

Array Basics

An **array** data structure (or simply "array") is a data structure consisting of a collection of elements (values or variables), each identified by one or more integer indices. (Source: WikiPedia.org)

Arrays are using for store similar data types grouping as a single unit.

Arrays must be declared in the same way variables are declared.

Example of declaring an array.

```
Dim myArray(100) As Integer
```

Note: This declares a 101 element array (since it starts at 0)

0	
1	
2	
3	
4	
5	
...	
99	
100	

You can also specify the values of array members when you declare the array.

```
Dim nums() As Integer = {0,1,2,3,4}
```

Erasing an Array

You can erase the contents of an array as follows ...

```
Erase myArray
```

Note: This assumes that you have previously declared the array `myArray`.

Two-Dimensional Arrays

Arrays can have more than one "dimension".
A two-dimensional array is similar to a matrix in math.

Example of declaring a two-dimensional array.

```
Dim myArray(5, 3) As Integer
```

Note: This declares a 24 element array. (ie. 6 by 4)

0,0	**0,1**	**0,2**	**0,3**
1,0	**1,1**	**1,2**	**1,3**
2,0	**2,1**	**2,2**	**2,3**
3,0	**3,2**	**3,2**	**3,3**
4,0	**4,1**	**4,2**	**4,3**
5,0	**5,1**	**5,2**	**5,3**

Similar to one-dimensional arrays, you can pre-assign values when you declare the array.

```
Dim rectArray(,) As Integer =
{{1, 2, 3}, {12, 13, 14}, {11, 10, 9}}
```

Note: Put this all on one line.

Multi-Dimensional Arrays

Arrays can also have more than two dimensions.

Arrays and Loops

Loops (for example: For-Next) are very useful for working with arrays.

The following code example illustrates a simple password program. The valid passwords are stored in an array, and when the user types something in TextBox1 and clicks Button1, the program determines if the user has entered a valid password.

In **General | Declarations** ...

```
Dim i As Integer
Dim password(4) As String
Dim passwordvalid As Boolean
```

In **Form | Load** ...

```
password(0) = "hello"
password(1) = "enter"
password(2) = "admin"
password(3) = "user"
password(4) = "access"
```

In **Button1 | Click** ...

```
If TextBox1.Text = "" Then Exit Sub

passwordvalid = False

For i = 0 To 4

    If TextBox1.Text = password(i) Then passwordvalid = True

Next

If passwordvalid = True Then

    MessageBox.Show("Valid Password.")

Else

    MessageBox.Show("Invalid Password.")

End If
```

Note: The line in blue could also be written as using the upper bounds and lower bounds commands ...

```
For i = password.lbounds() to password.ubounds()
```

... or also as ...

```
For i = 0 To password.Length - 1
```

Note: We use *Lenght - 1* since the lenght of the array is **5**, but the elements actually go from 0 to **4**.

The Bubble Sort

Elements stored in arrays can easily be sorted using a technique known as the **Bubble Sort**. The Bubble Sort is one of the most straight-forward of the numerous sort methods available, but it is also one of the least efficient. Nevertheless, if you want to sort a few hundred items, this method is one of the fastest available.

The bubble sort algorithm works by comparing adjactent array elements and interchanging (swapping) the ones that are out of order.

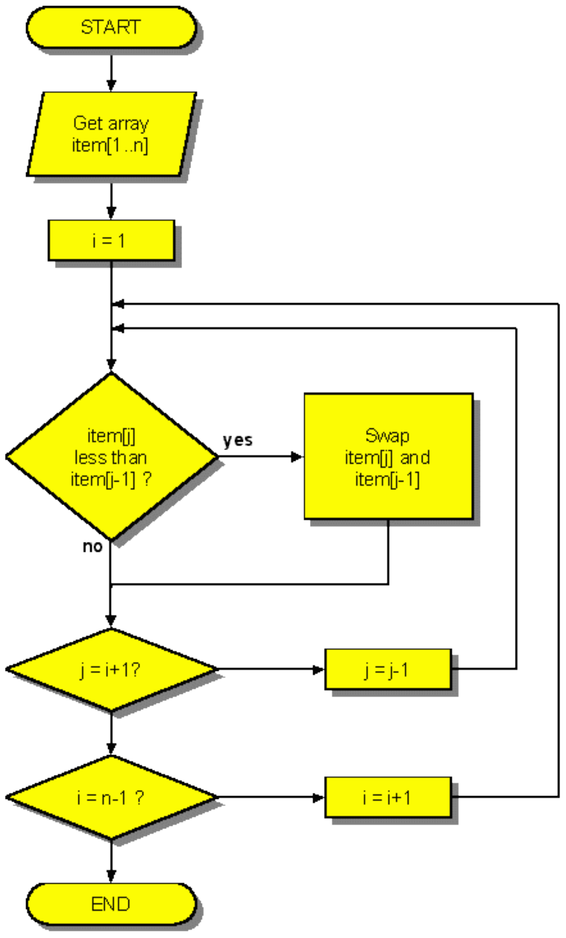
The example below assumes that you have an array called `nums` that has been declared as follows:

```
Dim nums(100) As Integer
```

The code also assumes that each element contains an integer value.

```
For i = nums.UBound() - 1 To 1 Step -1
    For j = 1 To i
        If nums(j) > nums(j + 1) Then
            temp = nums(j)
            nums(j) = nums(j + 1)
            nums(j + 1) = temp
        End If
    Next j
Next i
```

Flowchart for the Bubble Sort



Note: The flowchart below varies slightly from the algorithm discusses above.

Source: Edge Diagrammer, © PaceStar Software

Array Exercises

Name ______________________ Date ______________________

An array has been declared as follows ... `Dim Nums(9) As Integer`

You can assume that each of the ten entries contains an integer value between -100 and 100.

1. Using a For-Next loop, write the code which will add up all the individual entries of the array and store the total in an integer variable named `SubTotal`.

2. Write the code that will return the average of the numbers in the array.
   Note: this should only involve a small modification of the code from 1.

3. Write the code that will return both the smallest and largest entries in the array.